Submit (on Gradescope) a 3-page, single-spaced report in PDF format describing details regarding the steps you followed for developing the classifier for predicting the product review sentiments.  
Be sure to include the following in the report:  
1. Your identifier as registered on the miner website.  
2. Rank & accuracy score for your submission (at the time of writing the report).  
3. A detailed and clear description of your approach, and how and why you chose the parameters, features, distance measure, etc.  
4. Any graphs or tables illustrating key experiments you did in the process of choosing your final model.

From the beginning, I had planned to use a bag of words approach to organizing the provided training data by collecting all the relevant words in a list. I planned on merging all vectors if the sentiment of the reviews matched (positive reviews merged with other positive reviews, negative reviews merged with other negative reviews) and doing a frequency count instead of a binary appearance count. In technical terms, I created two python dictionaries (one for positive sentiment and one for negative) and mapped a frequency count (or degree as written in my code) to each word found in the training file based on the sentiment given in the training file.

At first attempt, my accuracy score was actually higher than I expected (at a 0.80). I was surprised because I submitted the results of a barebones approach to the assignment. In the first draft code, I considered all words (capitalization matters). I used a simple weight scheme to determine if a word from a review would be considered in the classification algorithm. The code would read a word from the review and it would find the ratio of positive over the negative counts from the training phase (inverted if the denominator were higher) so that the ratio would equate to a real number greater than 1 or less than -1, and the code would then sum up all of the word ratios in the review to see if the sum is a positive or negative result. This sort of variation of the K-nearest implementation takes in words that occurred in both training reviews and the current test review to do a 100% K-nearest analysis. This means that the test review would consider all of the training data (K would be some absurd number like 112,000 for both positive and negative). However, there would be weights attached to each of the words due to the ratio implementation.

I decided to expand my code by implementing feature subset selection. The simplest way I thought was to suppress any redundant words during training, and I found a nice list of stop words here: <https://countwordsfree.com/stopwords>. With this list, I was able to utilize feature subset selection by removing noise from the training subset. However, with each new attempt (adding more and more stop words), I found myself getting lower scores each time. This made me call back to the lecture when the professor explained that bag of words is strong because of its simplicity. I ended up reducing the list of stop words to only two words: “THE” and “A”. Here, I realized I made a simple mistake of including redundant words with different capitalizations. I quickly changed my code to ignore capitalization. This ended up stopping my score from dropping even further past a 0.49 (the last feature subset selection code attempt resulted in a 0.54). Because increasing the complexity of the preprocessing of inputs seemed to be worse for the algorithm, I decided that I had to create a better K-nearest algorithm approach instead of trying to decrease the dimensionality.